# 4 Integration von Deep Learning in Reinforcement Learning Verfahren am Beispiel von Pac-Man

Die Integration künstlicher neuronaler Netze, in Reinforcement Learning Verfahren bringt zunächst einige Hürden aus Sicht des Deep Learning mit sich, die berücksichtigt werden müssen. In der Regel brauchen künstliche neuronale Netze große Datensätze, bestehend aus Eingabe- und zugehörige Ausgabedaten, um ein Problem effizient lösen zu können *(vgl. Minh et a.l, 2013, S.1)*. In Reinforcement Learning Verfahren hingegen, ist die Idee, dass ein Agent durch Belohnungen (Rewards), in Form von natürlichen Zahlen, lernt Probleme zu lösen *(vgl. Sutton & Barto, 2018, S.6).* Die Belohnungen erhält der Agent durch die Umgebung, in der er sich befindet. Es kann vorkommen, dass ein Agent erst verspätet Belohnungen erhält, aus denen er lernen kann. Dies steht im Gegensatz zu überwachten Lern-Algorithmen, die zu jeder Eingabe, eine zu lernende Ausgabe besitzen *(vgl. Minh et al., 2013, S.1).* Des Weiteren verlangen Deep Learning Algorithmen, dass Eingaben in das künstliche neuronale Netz unabhängig voneinander sind, eine Eigenschaft, die bei Reinforcement Learning Problemen zu Schwierigkeiten führt, da Spielsituationen aufeinander aufbauend sind *(vgl. Minh et al., 2013, S.1).* Die im folgenden beschriebenen Verfahren lösen diese Probleme auf und ermöglichen die effiziente Nutzung von Deep Learning Algorithmen in Reinforcement Learning Verfahren.

## 4.1 Beispiel Pac-Man

Bei Pac-Man handelt es sich um ein Spiel, bei dem ein Agent (Pac-Man) durch ein Labyrinth navigiert und dabei möglichst viele Punkte („Dots“) sammelt. Dabei wird der Agent von Geistern, die als Gegenspieler agieren, verfolgt. Schafft es der Agent alle sich auf dem Spielfeld befindenden Punkte zu sammeln, hat er gewonnen. Kollidiert der Agent bei seinem Versuch alle Punkte zu sammeln mit einem Geist, hat er das Spiel verloren.

Um dieses Spiel als ein Reinforcement Learning Problem zu beschreiben, betrachten wir Pac-Man als unseren Agenten. Für jeden gefressenen Punkt bekommt der Agent eine Belohnung. Kollidiert der Agent vorher mit einem Geist und verliert das Spiel, erhält er eine (hohe) negative Belohnung. Die Umgebung ist hier das Labyrinth, indem sich der Agent befindet.

In unseren Versuchen haben wir einen 7x7 großes Spielfeld mit einem Geist als Gegenspieler gewählt. Der gewählte Geist hat die Eigenschaft, sich zufällig durch das Labyrinth zu bewegen. Das Ziel, unserer Versuche war es, dass der Agent selbständig lernt, das Spiel erfolgreich abzuschließen.

```
[['W', 'W', 'W', 'W', 'W', 'W', 'W'],
 ['W', 'P', 'D', 'D', 'D', 'D', 'W'],
 ['W', 'D', 'W', 'W', 'W', 'D', 'W'],
 ['W', 'D', 'W', 'D', 'D', 'D', 'W'],
 ['W', 'D', 'W', 'D', 'W', 'D', 'W'],
 ['W', 'D', 'D', 'D', 'D', 'G', 'W'],
 ['W', 'W', 'W', 'W', 'W', 'W', 'W']]
```

**Abbildung 14:** Grafische und textuelle Darstellung des Spiels

### 4.1.1 Preprocessing

Die Spieleumgebung, die wir für unsere Versuche verwendet haben, hat die aktuelle Spielsituation in Form einer textuellen Beschreibung an unseren Agenten übergeben (siehe ***Abbildung 14***). Hierbei steht W für „Wall“, P für „Pac-Man“, D für „Dot“ und G für „Ghost“.

In ersten Durchläufen haben wir probiert, die Verschiedenen Elemente der textuellen Wahrnehmung („W“, „P“, „D“, „G“) in unterschiedliche Zahlen zu kodieren. Die so entstandene Matrix wurde dann in einen Vektor umgeformt und in das künstliche neuronale Netz eingegeben. Das Ergebnis nach 5000 Episoden des Trainings ließen uns aber darauf schließen, dass dieser Ansatz nicht funktioniert. Der Agent bewegte sich nach dem Training weitestgehend zufällig durch das Labyrinth und schien dabei nicht gelernt zu haben. Wir vermuten, dass das künstliche neuronale Netz sich damit schwergetan hat, die Abhängigkeiten der einzelnen Felder voneinander zu verstehen, da durch die Vektoreingabe positionelle Strukturen des Spielfeldes verloren gingen.

In folgenden Versuchen haben wir ein Faltungsnetzwerk (Convolutional Neural Network) verwendet, das Eingaben durch Bilder ermöglicht *(vgl. Krizhevsky et al., 2012, S. 1097-1105)* und so die positionellen Eigenschaften des Spielfelds nicht verloren gehen. Hierfür mussten wir die textuelle Wahrnehmung unseres Agenten zuerst in ein Bild umwandeln. Dazu wurden die Bibliotheken PIL und Numpy verwendet.

```python
def draw_state(self, state):
    im = Image.new("RGB", (len(state)*12, len(state[0])*12))
    draw = ImageDraw.Draw(im)

    for x in range(0, len(state)):
        for y in range(0, len(state[x])):
            if state[y][x] == 'W':
                draw.rectangle((x*12, y*12, x*12+12, y*12+12), fill=(0, 0, 139))
            elif state[y][x] == 'P':
                draw.rectangle((x*12, y*12, x*12+12, y*12+12), fill=(255, 255 ,0))
            elif state[y][x] == 'G' or state[x][y] == 'GD':
                draw.rectangle((x*12, y*12, x*12+12, y*12+12), fill=(255, 0, 0))
            elif state[y][x] == 'D':
                draw.rectangle((x*12, y*12, x*12+12, y*12+12), fill=(255, 255, 255))
            elif state[y][x] == 'E':
                draw.rectangle((x*12, y*12, x*12+12, y*12+12), fill=(0, 0, 0))
    #im.show()
    im = np.array(im)
    im = np.reshape(im, (1, 84, 84, 3))

    return im
```

**Abbildung 15:** Funktion, die die textuelle Wahrnehmung in eine Grafik umwandelt

Bei der Umwandelung der textuellen Wahrnehmung in eine Grafik, haben wir uns an den in *(Minh et al., 2013, S.5)* benutzten Maßen (84x84 Pixel) orientiert.

Um die 7x7 große textuelle Wahrnehmung auf ein 84x84 Pixel großes Bild zu bringen, mussten wir jedes Textelement der Wahrnehmung in ein 12x12 Pixel großes Quadrat umwandeln.

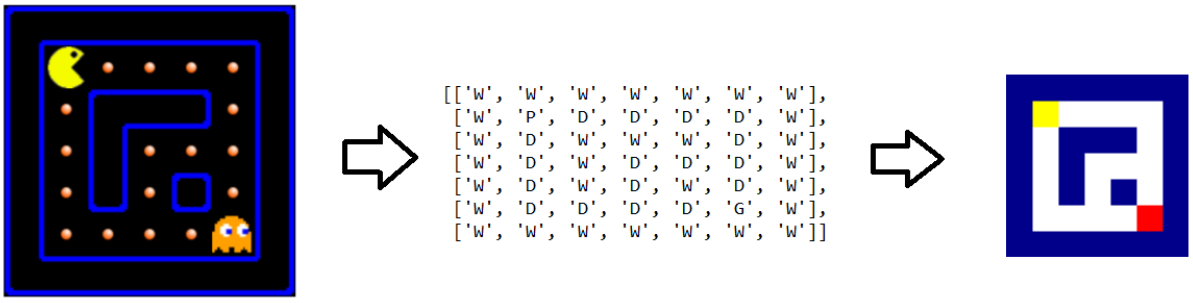


**Abbildung 16:** Pipeline von Spiel zu grafischer Wahrnehmung

Die so entstandenen Bilder wurden anschließend in Graustufenbilder (siehe ***Abbildung 17***) umgewandelt, um die Dimensionalität der Bilder von drei Farbkanälen auf einen Kanal zu reduzieren. Dies spart wertvolle Rechenkapazität ein und erleichtert die Berechnung für das künstliche neuronale Netz, da die Eingabe Dimensionalität des Netzes damit verringert wird *(vgl. Minh et al., 2013, S.5).*

Einen Einfluss auf die Ergebnisse hat die Umwandelung in Graustufenbilder nicht, da die genauen Farbwerte nicht relevant für den Agenten sind, um das Spiel zu erlernen.

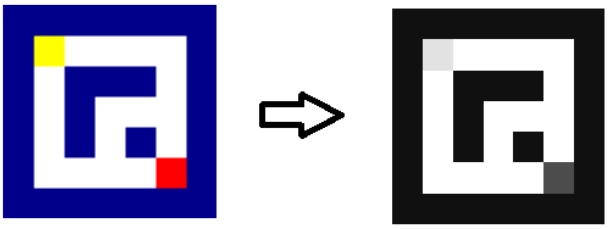

**Abbildung 17:** Reduzierung der Farbkanäle zu einem Graustufenkanal

Zusätzlich wurden immer zwei aufeinanderfolgende Bilder zusammen in einer Buffer (Framestack), analog zum FIFO Prinzip, gespeichert (siehe ***Abbildung 18***). Dies war nötig, damit das künstliche neuronale Netz den Spielkontext versteht, Bewegungen realisiert und nicht nur statische Ausschnitte des Spiels als Eingabe eingegeben bekommt.

**Abbildung 18:** Zwei aneinanderhängende Spielzustände als Framestack

Nach dem Neustart des Spiels wurde für vier Spielzüge die Initialbeobachtung übergeben, da die textuelle Wahrnehmung, nach Neustart, für die ersten vier Spielzüge Fehlerhaft ist.

## 4.2 Integration von Deep Q-Learning

Der in diesem Kapitel beschriebene Algorithmus bezieht sich auf den in *(Minh et al., 2015, S.529 – 533)* vorgestellten Algorithmus. *(Minh et al., 2015, S.529 – 533)* ist eine Erweiterung von *(Minh et al, 2013)*, wobei ein sogenanntes „Target-Network“ eingeführt wurde, welches für ein stabileres Verfahren zum Erlernen von Atari Spielen verholfen hat. Bei dem „Target-Network“ handelt es sich um ein künstliches neuronales Netz, das eine identische Struktur zu dem während des Spiels aktiv lernenden Netzes hat.

Durch die von uns verwendete Pac-Man Umgebung, fällt der angewandte Algorithmus in die Klasse der Modell-freien Algorithmen. Die Umgebung ist stochastisch, da die Aktionen des Gegenspielers – der Geist – nicht vorhergesagt werden können. Somit kann kein festes Modell der Umgebung erstellt werden, wodurch strategisches Planen ausgeschlossen wird *(vgl. Sutton & Barto, 2018, S.159ff)*.

Bei Deep Q-Learning werden die Wertigkeiten der unterschiedlichen Spielzustände (States) mit Hilfe eines künstlichen neuronalen Netzes (Deep Q-Network) erlernt. Genauer wird hierbei durch das künstliche neuronale Netz versucht die optimale „Action-Value“-Funktion zu approximieren. Das Approximieren geschieht wie auch bei klassischen Q-Learning Verfahren, die ohne ein künstliches neuronales Netz auskommen, mit Spielstand und die dazugehörigen Aktions-Paaren. Die Spielstände werden hierbei in das künstliche neuronale Netz gegeben und zu jeder möglich ausführbaren Aktion, ausgehend von der eingegebenen Spielsituation, wird ein Q-Value berechnet.

Als Policy (deutsch: Regel) wird die Aktion mit dem höchsten Q-Value ausgeführt. Da hierbei nicht direkt eine Policy erlernt wird, gehört dieses Verfahren zu der Klasse der „off-Policy“ Verfahren *(vgl. Sutton & Barto, 2018, S.131)*.

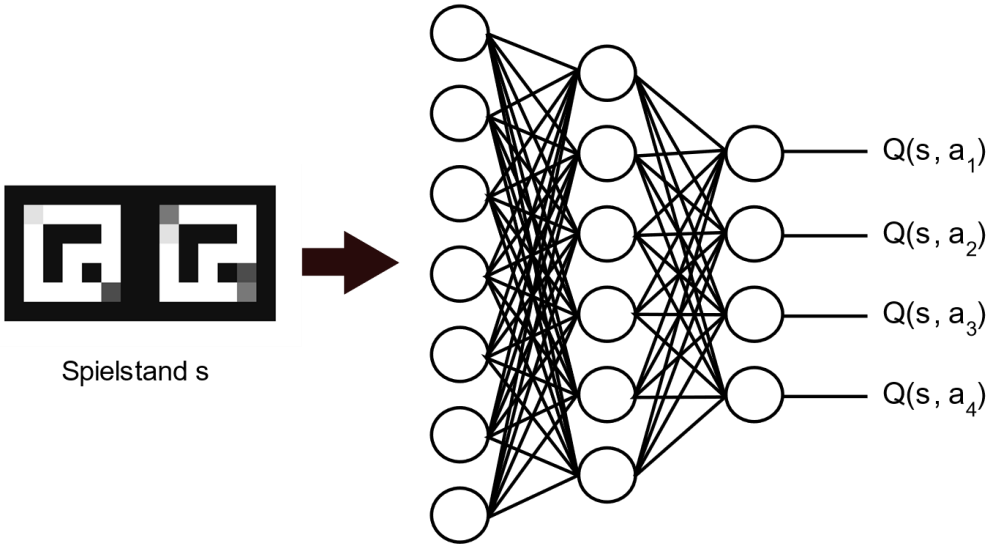


**Abbildung 19:** Deep Q-Network (am Beispiel eines Pac-Man States)

Um die o.g. Probleme des Deep Learnings zu beseitigen, werden Spielzüge als Tupel (Spielstand, gewählte Aktion, Belohnung, Spiel Termination, folge Spielstand) in einem Speicher (Replay-Memory) abgespeichert. Damit Eingaben in das Deep Q-Network unabhängig voneinander sind, werden aus dem Replay-Buffer zufällige Spielsituationen gezogen und in das Deep Q-Network zum Lernen eingegeben. Hierfür ist es notwendig, genügend Spielzüge im Replay-Memory abgespeichert zu haben. In überwachten Lernverfahren, des Deep Learnings, besteht der Datensatz immer aus Eingaben und dazugehörige festen Ausgaben. Beim Reinforcement Learning hingegen sind die Ausgaben, die Q-Values, keine festen Werte, sondern ändern sich während des Spielverlaufs. Mit Hilfe des Target-Networks werden Q-Values der folgenden Spielzustände berechnet und dann versucht im aktuellen Spielzustand zu approximieren.

## 4.3 Ablauf des Deep Q-Learnings

Zu Beginn eines Spiels (Episode) wird die Umgebung auf seinen Startzustand zurückgesetzt. Der Agent bekommt von der Umgebung eine Wahrnehmung und wählt eine Aktion, die durch den „ε-Greedy-Algorithmus“ bestimmt wird. Der „ε-Greedy-Algorithmus“ ist das genutzte Richtmaß zwischen Exploration und Exploitation. Durch die gewählte Aktion erhält der Agent eine neue Wahrnehmung, eine Belohnung und die Information, ob das Spiel terminiert ist. Der gesamte Vorgang wird dann im Replay-Memory als Tupel abgespeichert. Sind genügend Spielzüge im Replay-Memory gespeichert, fängt der Agent an das Deep Q-Network mit zufällig gewählten Tupeln, von Spielzügen, in einem Minibatch zu trainieren.

## 4.4 Deep Q-Network und Hyperparameter

Das eingesetzte künstliche neuronale Netz besteht aus einer Inputschicht, drei Faltungsschichten (*Conv2D*), einer „Fully Connected“ Schicht (*Dense*) und einer Ausgabeschicht. Die Inputschicht besitzt die Form (*84, 84, 2*), dies steht für die 84 mal 84 Pixel des Bildes und dass zwei aufeinanderfolgende Bilder übergeben werden. Die erste Faltungsschicht besitzt 16 Filter, mit einer Größe (*kernel size*) von

4 mal 4 und einer Schrittweite (*Strides*) von 2 und 2. Die zweite Faltungsschicht arbeitet mit 32 Filtern, der Größe 4 mal 4 und einer Schrittweite von 2 und 2. Die letzte Faltungsschicht besitzt, genau wie die vorherige Schicht, 32 Filter. Die Filtergröße ist in dieser Schicht 3 mal 3 und die Schrittweite 1 und 1.

Die letzte Faltungsschicht wird im Anschluss, mittels *GlobalAveragePooling2D*, in einer Form konvertiert, die einer *Dense* Schicht übergeben werden kann.

Die Dense Schicht besitzt 512 Neuronen. Die Ausgabeschicht, welche auch vom Typ Dense ist, besitzt vier Neuronen (für jede mögliche Aktion ein Neuron).

Nach jeder mit Neuronen gefüllten Schicht wird die Relu-Aktivierungsfunktion angewendet.

Zum Berechnen des Fehlers wurde die Huber-Funktion benutzt und als Optimierer der Adam-Optimierer.

Weitere Hyperparameter können der Tabelle ***Tabelle 1*** entnommen werden.

| **Hyperparameter** | **Wert** |
|---|---|
| Mini-Batch Größe | 32 |
| Replay-Memory Größe | 100000 |
| Training Start | 1000 |
| Discount Faktor | 0,995 |
| Lernrate | 0,00025 |
| ε-Start | 1 |
| ε -Ende | 0,01 |
| ε-Schritte | 100000 |
| ε-Schritt | ε -Aktuell - ε -Ende / ε-Schritte |
| Update Target-Network Schritte | 100 |

**Tabelle 1**: verwendete Hyperparameter

Für jeden gesammelten Punkt bekommt der Agent eine Belohnung von 5. Schafft der Agent es nicht, das Spiel abzuschließen, werden ihm von seiner aktuellen Belohnung 120 abgezogen. Damit der Agent lernt, zügig, ohne unnötige Aktionen auszuführen, das Spiel abzuschließen, erhält er für jede gewählte Aktion, bei der er keinen Punkt einsammelt, eine negative Belohnung von -1.

## 4.5 Resultat

Trainiert wurden 15000 Episoden, das entsprach dem Zeitraum von ungefähr einer Woche. Während des Trainings ließ sich beobachten, dass der Agent zuerst lernte, den äußeren Ring, der Umgebung, erfolgreich zu umlaufen und alle Punkte einzusammeln.

Schwierig tat sich der Agent damit, den mittleren Teil (siehe ***Abbildung 20***) zu erlernen. Dies kann die Folge verschiedener Ursachen sein. Zum einen durchläuft der Agent, zu Beginn des Lernprozesses, zufällig die Umgebung, um so ausreichend Spielzüge für den Replay-Speicher zu sammeln. Es ist unwahrscheinlich, dass der Agent, während der Exploration, eine Aktion wählt, um den Zugang zur

Spielfeld Mitte zu betreten. So gelangen nur wenige Tupel, die genau diese Aktion beschreiben, in den Replay-Speicher. Hinzu kommt die Größe des Replay-Speichers. Da sich vermutlich nur wenige Ausschnitte im Speicher befinden, bei denen der Agent die Spielfeldmitte durchläuft, müssen genau diese zufällig gezogen werden, sonst lernt der Agent nicht, in der Spielfeldmitte richtig zu agieren. Die Wahl eines kleineren Replay-Speichers könnte dieses Problem eindämmen.

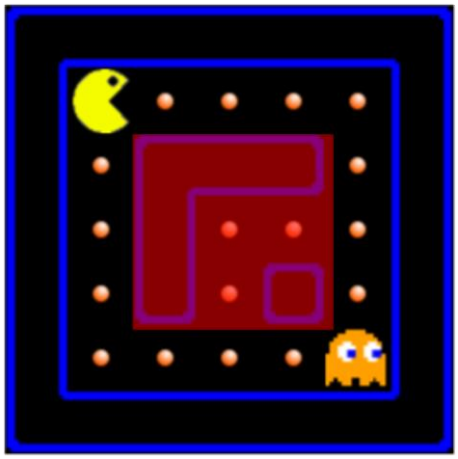

**Abbildung 20:** Problemzone Spielfeldmitte

Ein weiteres Problem bestand darin, dass der Agent seinen Gegenspieler, den Geist, nicht als Gefahr einstufte. Dies kann daran liegen, dass der Geist sich zufällig bewegt und der Agent das Verhalten des Geistes nicht vollständig verstehen konnte. Auch dies kann an dem zu großen Replay-Speicher liegen, da der Agent auch Runden spielte, in denen er dem Geist auswich.

Nach den 15000 Episoden hat der Agent das Spiel noch nicht vollständig erlernt, er wählte noch zu häufig Aktionen, die in einer negativen Belohnung resultierten und er konnte dem Gegenspieler nur selten ausweichen.

Diese Fehler reduzierten sich mit der Zeit immer stärker, sodass wir vermuten, dass es dem Agenten möglich ist, bei weiterem Training und mit kleinerem Replay-Speicher, das Spiel vollständig erfolgreich zu erlernen.

# Quellenverzeichnis


(Mnih et al., 2013)
V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wiestra, M. Riedmiller; Playing Atari with Deep Reinforcement Learning; https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf; 2013

(Mnih et al., 2015)
V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis: Human-level control through deep reinforcement learning; Nature, Seiten 529-533; 2015

(Krizhevsky et al., 2012)
A. Krizhevsky, I. Sutskever, G. Hinton; Imagenet classification with deep convolutional neural networks; Advances in neural information processing systems, Seiten 1097-1105; 2012

(Butler, 1863)
C. Butler; Darwin Among the Machines; The Press, Christchurch; 1863

(Döbel et al., 2018)
I. Döbel, M. Leis, M. Vogelsang, J. Welz, D. Neustroev, H. Petzka, A. Riemer; Maschinelles Lernen. Eine Analyse zu Kompetenzen, Forschung und Anwendung; Fraunhofer-Gesellschaft, München; 2018

(François-Lavet et al., 2018)
V. François-Lavet, P. Henderson, R. Islam, M. Bellemare, J. Pineau; An Introduction to Deep Reinforcement Learning; In Foundations and Trends in Machine Learning Vol. 11 No. 3-4; 2018

(Sutton & Barto, 2018)
R. Sutton und A. Barto; Reinforcement Learning, An Introduction; The MIT Press, Cambridge; 2018

(van Hasselt, 2012)
H. van Hasselt; Double Q-learning; Advances in neural information processing systems, Seiten 2613-2621; 2012

(van Hasselt et al., 2016)
H. van Hasselt, A. Guez, D. Silver; Deep Reinforcement Learning with Double Q-Learning; Association for the Advancement of Artificial Intelligence (AAI); 2016

(Watkins, 1989)
C. Watkins; Learning from delayed rewards; Cambridge; 1989

(Goodfellow et al., 2018)
I. Goodfellow, Y. Bengio, A. Courville; Deep Learning: Das umfassende Handbuch; mitp-Verlag, Bonn; 2018

(Zeiler & Fergus, 2014)
M. Zeiler und R. Fergus; Visualizing and Understanding Convolutional Networks; 2014

(Géron, 2018)
A. Géron; Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow; O'Reilly; 2018

(Rosenblatt, 1957)
F. Rosenblatt; The Perceptron — A Perceiving and Recognizing Automaton; 1957

(Strecker, 1997)
S. Strecker; Künstliche Neuronale Netze – Aufbau und Funktionsweise; http://geb.uni-giessen.de/geb/volltexte/2004/1697/pdf/Apap_WI_1997_10.pdf; 1997